Asynchronous Sockets

Tran Giang Son, tran-giang.son@usth.edu.vn

ICT Department, USTH

Contents

What & Why

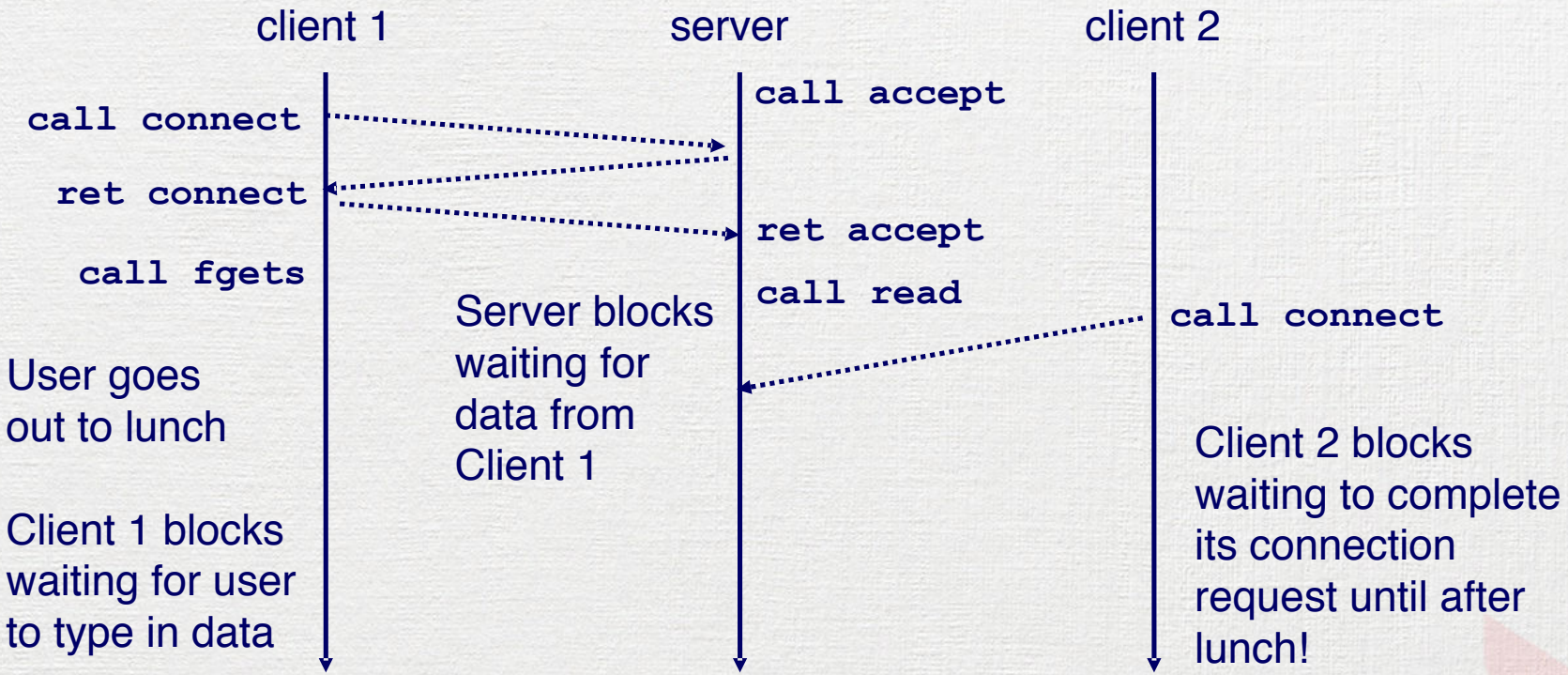
What?

- By default: blocking, e.g.
  - `accept()` only returns when there's an incoming connection
  - `read()`/`recv()` only return when there's some data
  - `write()`/`send()` only return when data is sucessfully sent
- Nonblocking: calls to network functions return to caller immediately

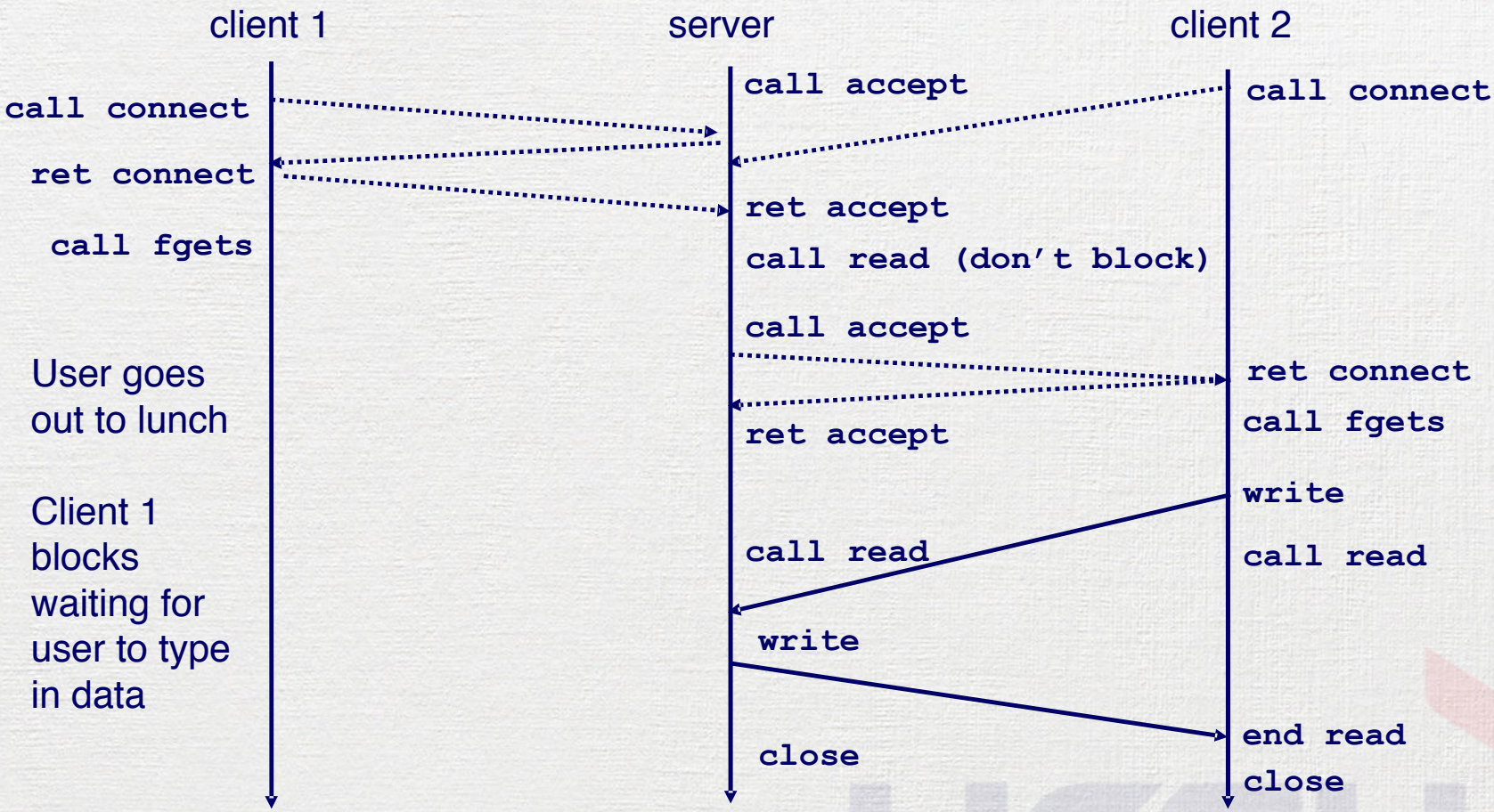
Why?

- With blocking socket
  - Server can only serve 1 client at anytime
  - Needs to take turn
  - No timeout

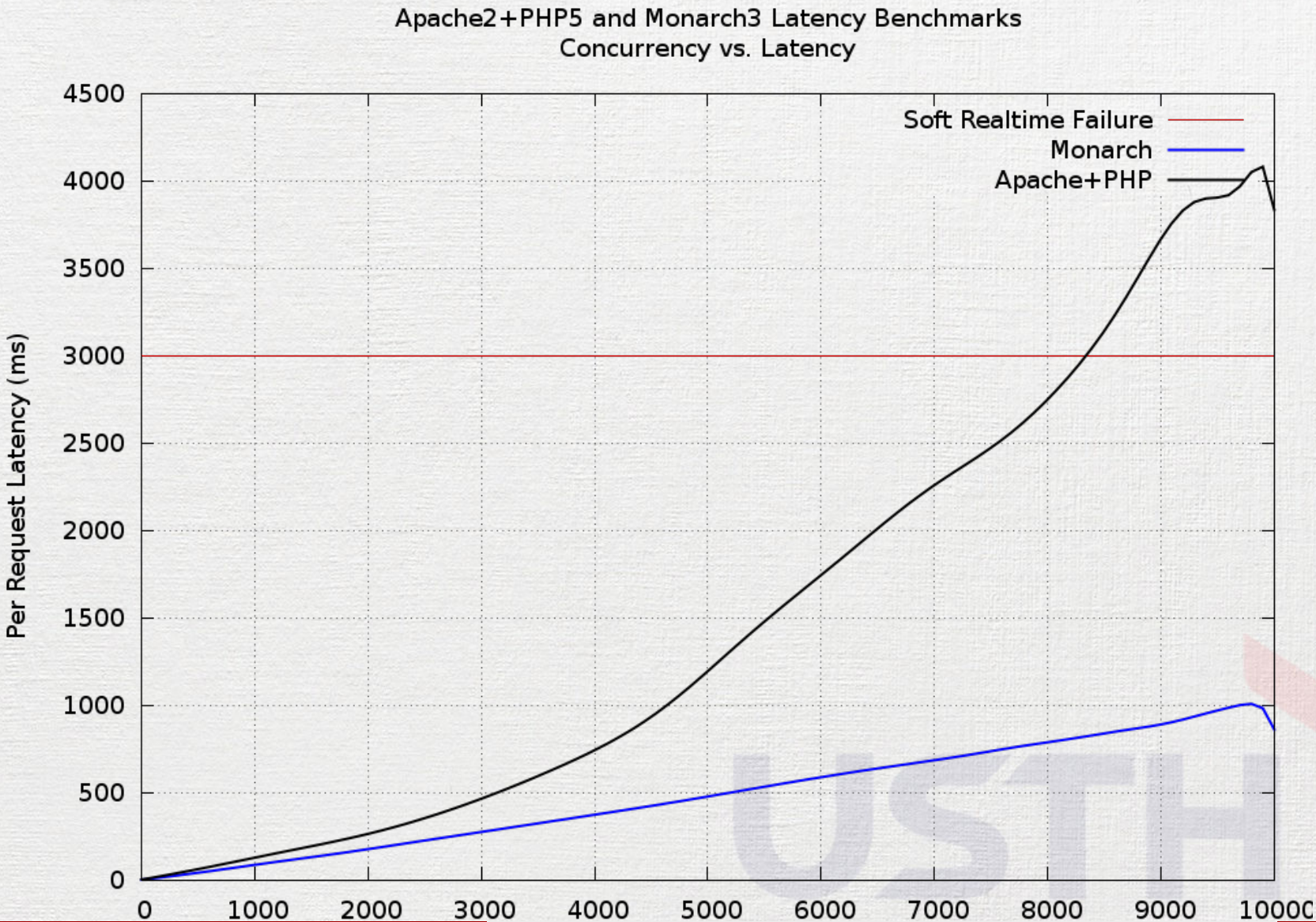
Why?


client 1
server
client 2
call connect
call accept
ret connect
ret accept
call fgets
call read
call connect
Server blocks waiting for data from Client 1
User goes out to lunch
Client 1 blocks waiting for user to type in data
Client 2 blocks waiting to complete its connection request until after lunch!

Why?


client 1
server
client 2
call connect
call accept
call connect
ret connect
ret accept
call fgets
call read (don't block)
call accept
ret connect
User goes out to lunch
ret accept
call fgets
write
Client 1 blocks waiting for user to type in data
call read
call read
write
end read
close
close

Why?


Apache2+PHP5 and Monarch3 Latency Benchmarks
Concurrency vs. Latency
Per Request Latency (ms)
Soft Realtime Failure
Monarch
Apache+PHP
0
500
1000
1500
2000
2500
3000
3500
4000
4500
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000

What we need

- A single server that supports multiple client concurrently:
  - accepts multiple connections
  - receives messages from all connected clients
  - sends message from STDIN to all clients

Nonblocking Sockets

What?

- Nonblocking operations
  - `connect()`
  - `accept()`
  - `read()` / `write()`
  - `send()` / `recv()`

What?

- Looks great!
- . . . but. . .
  - Complication
  - Maintenance

How to use it?

- Allow reusing address
- Enable nonblocking option
- Restructure server & client
- Profit.

Reusing address

```
int setsockopt(int socket, int level,
    int option_name,
    const void *option_value, socklen_t option_len);
```

- **Set Sock**et **Opt**ions
- `socket`: the file descriptor returned by `socket()`
- `level`: protocol level
- `option_name`: the option that we need to adjust
- `option_value`: value for that option
- `option_len`: its length

Reusing address

Name	Meaning
`SO_DEBUG`	recording of debugging information
`SO_REUSEADDR`	**local address reuse**
`SO_REUSEPORT`	duplicate address and port bindings
`SO_KEEPALIVE`	keep connections alive
`SO_DONTROUTE`	routing bypass for outgoing messages
`SO_BROADCAST`	permission to transmit broadcast messages
`SO_SNDBUF`	set buffer size for output
`SO_RCVBUF`	set buffer size for input
`SO_SNDTIMEO`	set timeout value for output
`SO_RCVTIMEO`	set timeout value for input
`SO_TYPE`	get the type of the socket (get only)
`SO_ERROR`	get and clear error on the socket (get only)
`SO_NOSIGPIPE`	do not generate SIGPIPE, instead return EPIPE
`SO_LINGER_SEC`	linger on close if data present with timeout in seconds
...	...

Reusing address

Example:

```
setsockopt(sockfd, SOL_SOCKET,
    SO_REUSEADDR, &(int){ 1 },
    sizeof(int));
```

Enable nonblocking option

- `fcntl(int fd, int command, int value)`: file control
  - `F_GETFL`
  - `F_SETFL`
- `O_NONBLOCK`
- Example:

```
int fl = fcntl(fd, F_GETFL, 0);
fl |= O_NONBLOCK;
fcntl(fd, F_SETFL, fl);
```

Restructure server & client

Blocking Server

```
socket()...
bind()...
listen()...
while (1) {
  clientfd = accept();
  while (1) {
    read()...
    printf()...
    scanf()...
    write()...
  }
}
close()...
```

Non-blocking Server

```
socket()...
setsockopt()...    // reuse address
fcntl()...         // nonblocking
bind()...
listen()...
while (1) {
  clientfd = accept();
  if (clientfd > 0) {
    fcntl()... // nonblocking client
    while (1) {
      if (read()... > 0) printf()...
      if (poll()...) {
        scanf()...
        write()...
      }
    }
  }
```

Restructure server & client

Blocking Client

```
socket()...
gethostbyname()...
connect()...
while (1) {
    scanf()...
    write()...
    read()...
    printf()...
}
close()...
```

Non-blocking Client

```
socket()...
gethostbyname()...
connect()...
setsockopt()...    // reuse address
fcntl()...         // nonblocking
while (1) {
  if (read()... > 0) printf()...
  if (poll()...) {
    scanf()...
    write()...
  }
}
```

Practical Work 8: Nonblocking System

- Copy your client and server code from 7^{th} practical work to
  - « 08.practical.work.server.nonblock.c »
  - « 08.practical.work.client.nonblock.c »
  - Improve server and client: nonblocking sockets
- Test the system between your laptop and VPS
- Do you see any problem with your server & client?
- Push your C programs to corresponding forked Github repository